UFLA – Universidade Federal de Lavras
Departamento de Ciência da Computação
COM167 – Teoria da Computação
Prof. Rudini Sampaio
Monitor: Rodrigo Pereira dos Santos

Segunda Lista de Exercícios – **2005/1**

.....

Exercício 1 Prove que a linguagem F_{TM} = {<M> | M.T. M é tal que L(M) é finita} é indecidível.

RESPOSTA =

Por contradição, assuma que existe uma M.T. R que decide F_{TM}.

Vamos construir uma M.T. S, baseada em R, que decide A_{TM} = {<M,w> | M aceita w}, que é indecidível.

Considere a M.T. M_1, cuja descrição depende de M e w.

M_1 = "Com entrada x:
1. Se x = 0 ou x = 1, ACEITE.
2. Simule M com entrada w.
3. Se M aceita w, REJEITE.
   Senão, ACEITE."

S = "Com entrada <M,w>:
1. Construa M_1 baseado em M e w.
2. Simule R com entrada <M_1>.
3. Se R aceita, ACEITE.
   Senão, REJEITE."

Logo, S decide A_{TM}. Contradição, pois A_{TM} é indecidível. Logo, R não existe e F_{TM} é indecidível.

■

.....

Exercício 2 Usando mais de 10 linhas, explique o PCP, a prova de sua indecidibilidade (sem muitos detalhes técnicos) e porque ele não pode ser classificado em classes de complexidade de tempo, como P e NP.

RESPOSTA =

O PCP (*Post Correspondence Problem*) é um exemplo de problema indecidível relativo à manipulação de *strings*. Pode ser descrito como um tipo de quebra-cabeça, uma coleção de dominós, cada qual contendo duas *strings*, uma de cada lado. A tarefa é fazer uma lista destes dominós (repetições permitidas) tal que a *string* superior seja igual à *string* inferior. Esta lista é chamada casamento. Assim, o PCP consiste em determinar se uma coleção tem um casamento. Devido à equivalência entre problema e linguagem, pode-se definir o PCP como uma linguagem: PCP = {<P> | P é uma instância do PCP com um casamento}. A prova de que PCP é indecidível consiste em uma contradição, assumindo que uma M.T. R decide PCP e construindo uma M.T. S, baseada em R, que decide A_{TM} = {<M,w> | M aceita w}. Logo, deve-se mostrar que a partir de uma M.T. M e uma entrada w, pode-se construir uma instância P, onde um casamento seria uma computação aceita em M com w. Para simplificar, seja MPCP uma instância modificada do PCP, tal que tenha duas restrições: uma computação de M sobre w nunca tenta mover o cursor à esquerda do início da fita; um dominó específico inicia o casamento. Como S constrói uma instância do PCP P que tem um casamento se e somente se M aceita w, construiremos primeiro uma instância P' de MPCP, descrita em 7 partes que acompanham um aspecto particular da simulação de M com w. Com isso, devemos lembrar que

MPCP difere do PCP por começar o casamento com um primeiro dominó específico. Se considerarmos P' como uma instância do PCP, ao invés de MPCP, tem-se um casamento. Assim, deve-se converter P' para P, de modo a utilizar P, que é uma instância do PCP, usando regras e "peças" específicas. Em síntese, com o fim da construção, S decidiria A_{TM}, que é indecidível. Contradição. Logo, nem S, nem R existem e PCP é indecidível. Por esse motivo, ele não pode ser classificado em classes de complexidade de tempo, uma vez que estas estão associadas a linguagens decidíveis (recursivas).

.....

Exercício 3 Com respeito a relação $\leq_m$, responda aos itens abaixo:

2.1.[5.4 Sipser] Se $A \leq_m B$ e B é uma linguagem regular, isso implica que A é regular?

RESPOSTA =

Não. Vamos mostrar isso, através de um contra-exemplo, construindo uma função de redução de uma linguagem não-regular A para uma linguagem regular B.

Tome $A = \{0^n1^n \mid n \geq 0\}$, que não é regular, e $B = \{0^n \mid n \geq 0\}$, que é regular (ambos provados na disciplina Linguagens Formais e Autômatos)

F = "Com entrada w:

1. Para cada 0 e 1 que aparecem juntos na palavra w, retire-os de w e coloque um 0 na palavra de saída. Faça isso até que $w = \varepsilon$.
2. Retorne a palavra de saída."

Concluímos que existe uma função de redução de A para B e $A \leq_m B$. Então, se B é regular, isso não implica que A é regular.

■

2.2.[5.5 Sipser] Mostre que A_{TM} não é redutível ao E_{TM}.

RESPOSTA =

Por contradição, suponha que $A_{TM} \leq_m E_{TM}$. Isso implica que $\neg A_{TM} \leq_m \neg E_{TM}$, por definição ($\neg A_{TM}$ e $\neg E_{TM}$ são complemento ou negação de A_{TM} e E_{TM}, respectivamente).

Vamos provar que $\neg E_{TM}$ é recursivamente enumerável. Para isso, vamos construir uma M.T. N que reconhece $\neg E_{TM}$.

Primeiramente, seja M uma M.T. tal que M aceita uma palavra $w \in \sum^*$ e $\sum^* \neq \emptyset$.

N = "Ignore a entrada:

1. Para i = 1, 2, 3, ...

1.1 Simule M com entrada s_i, onde s_1, s_2, ... é cada um dos elementos de $\sum^*$ em ordem lexicográfica.

1.2 Se M aceita, ACEITE."

Logo, $\neg E_{TM}$ é recursivamente enumerável.

Como $A_{TM} \leq_m E_{TM}$ e E_{TM} é recursivamente enumerável, então A_{TM} é recursivamente enumerável, por teorema. Contradição, pois $\neg A_{TM}$ não é recursivamente enumerável (Corolário 4.17).

Dessa forma, A_{TM} não é redutível polinomialmente a E_{TM}.

■

2.3.[5.6 Sipser] Mostre que $\leq_m$ é uma relação transitiva.

RESPOSTA =

Sejam A, B e C três linguagens, tal que $A \leq_m B$ e $B \leq_m C$.

Sejam f e g funções de redução de A para B e de B para C, respectivamente.

Então, w Є A ↔ f(w) Є B e y Є B ↔ g(y) Є C.

Logo, w Є A ↔ f(w) Є B. Se f(w) Є B, f(w) Є B ↔ g(f(w)) Є C.

Dessa forma, conclui-se que existe uma função de redução de A para C, que é g(f(w)) ou $g \circ f$. Logo, $\leq_m$ é uma relação transitiva.

■

2.4.[5.7 Sipser] Mostre que se $A \leq_m \bar{A}$ e A é recursivamente enumerável, então A é recursiva.

RESPOSTA =

Como $A \leq_m \bar{A}$, por definição, $\bar{A} \leq_m A$.

Como A é recursivamente enumerável, pelo Teorema 5.22, temos que $\bar{A}$ é recursivamente enumerável. Pelo teorema 4.16, uma linguagem é recursiva se e somente se ela e seu complemento são recursivamente enumeráveis. Logo, A é recursiva.

■

2.5.[5.9 Sipser] Mostre que todas linguagens recursivamente enumeráveis são redutíveis ao A_{TM}.

RESPOSTA =

Seja L uma linguagem recursivamente enumerável qualquer. Logo existe uma M.T. M que a reconhece.

Vamos provar que para todo L, $L \leq_m A_{TM}$. Vamos construir uma função de redução f: $L \rightarrow A_{TM}$ tal que w Є L ↔ f(w) Є A_{TM}.

Seja a função f(w) = <M,w>, tal que M aceita w.

Como a função é computável e w Є L ↔ f(w) Є A_{TM}, temos que todas as linguagens recursivamente enumeráveis são redutíveis ao A_{TM}.

■

2.6.[5.11 Sipser] Mostre um exemplo de uma linguagem indecidível A tal que $A \leq_m \bar{A}$.

RESPOSTA =

Considere as linguagens EQ_{TM} e $\neg EQ_{TM}$, que não são recursivamente enumeráveis, pelo teorema 5.24 ($\neg EQ_{TM}$ é complemento ou negação de EQ_{TM}). Como EQ_{TM} e $\neg EQ_{TM}$ não são recursivamente enumeráveis, também não são recursivas (decididas). Logo, são indecidíveis.

Vamos provar que $EQ_{TM} \leq_m \neg EQ_{TM}$.

Vamos construir uma função de redução f: $EQ_{TM} \leq_m \neg EQ_{TM}$, tal que w Є EQ_{TM} ↔ f(w) Є $\neg EQ_{TM}$.

Seja f(<M_1,M_2>) = <M_1,M_3> tal que $L(M_1) = L(M_2)$ e $L(M_1) \neq L(M_3)$.

F = "Com entrada <M_1,M_2>:

1. Constua a M.T. M_3 tal que M_3 aceita quando M_2 rejeita, e M_3 rejeita quando M_2 aceita, ou seja, $L(M_3) = \neg L(M_2)$.
2. Devolva <M_1,M_3>."

Logo, como F é computável e <M_1,M_2> Є EQ_{TM} ↔ $L(M_1) = L(M_2) \neq L(M_3)$ ↔ <M_1,M_3> Є $\neg EQ_{TM}$. Logo, $EQ_{TM} \leq_m \neg EQ_{TM}$.

■

.....

Exercício 4 [7.6 Sipser] Mostre que a classe de linguagens ***P*** é fechada sob as operações de:

RESPOSTA =

Sejam L_1 e L_2 linguagens, tais que existem M.T.'s determinísticas M_1 e M_2 que decidem L_1 e L_2, respectivamente, em tempo polinomial. Então, $L_1, L_2 \in P$. Sejam f(n) e g(n) funções de complexidade de tempo para as M.T.'s M_1 e M_2, respectivamente.

a. União

Vamos construir uma M.T. determinística M_3 que decide a linguagem $L_3 = L_1 \cup L_2$.

M_3 = "Com entrada w:
1. Simule M_1 com entrada w.
2. Simule M_2 com entrada w.
3. Se M_1 ou M_2 aceitam, ACEITE.
   Senão, REJEITE."

A complexidade de tempo de M_3 é f(n) + g(n), que é polinomial. Então, M_3 decide L_3 em tempo polinomial. Logo, $L_3 \in P$ e a classe de linguagens P é fechada sob a operação de união.

■

b. Intersecção

Vamos construir uma M.T. determinística M_4 que decide a linguagem $L_4 = L_1 \cap L_2$.

M_4 = "Com entrada w:
1. Simule M_1 com entrada w.
2. Simule M_2 com entrada w.
3. Se M_1 e M_2 aceitam, ACEITE.
   Senão, REJEITE."

A complexidade de tempo de M_4 é f(n) + g(n), que é polinomial. Então, M_4 decide L_4 em tempo polinomial. Logo, $L_4 \in P$ e a classe de linguagens P é fechada sob a operação de interseção.

■

c. Concatenação

Vamos construir uma M.T. determinística M_5 que decide a linguagem $L_5 = L_1 \bullet L_2$.

M_5 = "Com entrada $w = w_1w_2w_3...w_n$, onde w_i é cada um dos caracteres de w:
1. Para i de 0 até n, faça:
1.1 Simule M_1 com entrada $w_1w_2...w_i$.
1.2 Simule M_2 com entrada $w_{i+1}...w_n$.
1.3 Se M_1 e M_2 aceitam, ACEITE.
   Senão, REJEITE."

A complexidade de tempo de M_5 é (n + 1)*[f(n) + g(n)], que é polinomial. Então, M_5 decide L_5 em tempo polinomial. Logo, $L_5 \in P$ e a classe de linguagens P é fechada sob a operação de concatenação.

■

d. Estrela

Vamos construir uma M.T. determinística M_6 que decide a linguagem $L_6 = L_1^*$.

M_6 = "Com entrada $w = w_1w_2w_3...w_n$, onde w_i é cada um dos caracteres de w:

1. Se w = ε, ACEITE.
2. Senão, para i de 0 até n faça:

2.1 Simule M_1 com entrada $w_1w_2...w_i$.

2.2 Simule M_6 com entrada $w_{i+1}...w_n$.

2.3 Se M_1 e M_6 aceitam, ACEITE.

3. REJEITE."

A complexidade de tempo de M_6 é n*f(n), que é polinomial. Então, M_6 decide L_6 em tempo polinomial. Logo, $L_6 \in P$ e a classe de linguagens P é fechada sob a operação estrela.

■

e. Complementação

Vamos construir uma M.T. determinística M_7 que decide a linguagem $L_7 = \neg L_1$ (complemento ou negação de L_1).

M_7 = "Com entrada w:

1. Simule M_1 com entrada w.
2. Se M_1 aceita, REJEITE.
   Senão, ACEITE."

A complexidade de tempo de M_7 é f(n), que é polinomial. Então, M_7 decide L_7 em tempo polinomial. Logo, $L_7 \in P$ e a classe de linguagens P é fechada sob a operação complemento.

■

.....

Exercício 5 [7.7 Sipser] Mostre que a classe de linguagens ***NP*** é fechada sob as operações de:

RESPOSTA =

Sejam L_1 e L_2 linguagens, tais que existem M.T.'s não-determinísticas M_1 e M_2 que decidem L1 e L2, respectivamente, em tempo polinomial. Então, $L_1, L_2 \in NP$. Sejam f(n) e g(n) funções de complexidade de tempo para as M.T.'s M_1 e M_2, respectivamente.

a. União

Vamos construir uma M.T. não-determinística M_3 que decide a linguagem $L_3 = L_1 \cup L_2$.

M_3 = "Com entrada w:

1. Simule M_1 com entrada w.
2. Simule M_2 com entrada w.
3. Se M_1 ou M_2 aceitam, ACEITE.
   Senão, REJEITE."

A complexidade de tempo de M_3 é f(n) + g(n), que é polinomial. Então, M_3 decide L_3 em tempo polinomial. Logo, $L_3 \in NP$ e a classe de linguagens NP é fechada sob a operação de união.

■

b. Intersecção

Vamos construir uma M.T. não-determinística M_4 que decide a linguagem $L_4 = L_1 \cap L_2$.

M_4 = "Com entrada w:

1. Simule M_1 com entrada w.

2. Simule M_2 com entrada w.
3. Se M_1 e M_2 aceitam, ACEITE.
   Senão, REJEITE."

A complexidade de tempo de M_4 é f(n) + g(n), que é polinomial. Então, M_4 decide L_4 em tempo polinomial. Logo, $L_4 \in$ NP e a classe de linguagens NP é fechada sob a operação de interseção.

■

c. Concatenação

Vamos construir uma M.T. não-determinística M_5 que decide a linguagem $L_5 = L_1 \bullet L_2$.

M_5 = "Com entrada $w = w_1w_2w_3...w_n$, onde w_i é cada um dos caracteres de w:

1. Para i de 0 até n, faça:
1.1 Simule M_1 com entrada $w_1w_2...w_i$.
1.2 Simule M_2 com entrada $w_{i+1}...w_n$.
1.3 Se M_1 e M_2 aceitam, ACEITE.
   Senão, REJEITE."

A complexidade de tempo de M_5 é (n + 1)*[f(n) + g(n)], que é polinomial. Então, M_5 decide L_5 em tempo polinomial. Logo, $L_5 \in$ NP e a classe de linguagens NP é fechada sob a operação de concatenação.

■

d. Estrela

Vamos construir uma M.T. não-determinística M_6 que decide a linguagem $L_6 = L_1^*$.

M_6 = "Com entrada $w = w_1w_2w_3...w_n$, onde w_i é cada um dos caracteres de w:

1. Se $w = \varepsilon$, ACEITE.
2. Senão, para i de 0 até n faça:
2.1 Simule M_1 com entrada $w_1w_2...w_i$.
2.2 Simule M_6 com entrada $w_{i+1}...w_n$.
2.3 Se M_1 e M_6 aceitam, ACEITE.
3. REJEITE."

A complexidade de tempo de M_6 é n*f(n), que é polinomial. Então, M_6 decide L_6 em tempo polinomial. Logo, $L_6 \in$ NP e a classe de linguagens NP é fechada sob a operação estrela.

■

.....

Exercício 6 Prove que as linguagens abaixo estão em **NP**:

RESPOSTA =

Pela definição da classe de problemas NP, um problema é NP se existe algum verificador polinomial para a linguagem. Vamos construir verificadores polinomiais para as linguagens abaixo:

a. 3SAT = { <Φ> | Φ uma fórmula 3CNF satisfatível }

Verificador polinomial da linguagem 3SAT:

M_1 = "Com entrada <Φ,C>, onde Φ é uma fórmula qualquer. Através de propriedades de operações lógicas, pode ser transformada para a 3CNF. C é um conjunto de valores a serem atribuídos a cada uma das variáveis de Φ.

1. Atribua cada um dos valores pertencentes a C à respectiva variável pertencente a Φ.
2. Se essa atribuição de C faz com que Φ seja verdadeiro (ou seja, "1"), ACEITE.
3. Senão, REJEITE."

Como a atribuição de valores às variáveis e o cálculo do resultado são operações realizadas em tempo polinomial, o verificador é polinomial. Logo, 3SAT∈ NP.

■

b. VERT-COLOR = { <G,k> | G tem uma coloração própria nos vértices com k cores}

Verificador polinomial da linguagem VERT-COLOR:

M_2 = "Com entrada <G,K,C>, onde G é um grafo, V(G) é seu conjunto de vértices e E(G) é seu conjunto de arestas. C é um conjunto de duplas (a,b), onde a ∈ V e b é uma cor, e C deve ser verificado.

1. Para todo u ∈ V(G), para todo v ∈ V(G) e u ≠ v, faça:
1.1 Se (u,v) ∈ E(G) e a cor de u e a cor de v são iguais, REJEITE.
2. Se o número de cores diferentes for menor ou igual a k, REJEITE.
3. REJEITE."

Como a realização de testes e do *loop* (1) são operações realizadas em tempo polinomial, o verificador é polinomial. Logo, VERT-COVER ∈ NP.

■

c. HAM-CYCLE = { <G> | G possui um ciclo hamiltoniano }

Verificador polinomial da linguagem HAM-CYCLE:

M_3 = "Com entrada <G,C>, onde G é um grafo, V(G) é seu conjunto de vértices e E(G) é seu conjunto de arestas. C é uma permutação na qual seus elementos são vértices pertencentes ao grafo G; esse candidato a certificado deve ser testado.

1. Se a aresta (C_n,C_1) não pertence a E(G), REJEITE.
2. Senão, para i de 1 até (n – 1), faça:
2.1 Verifique se a aresta (C_i,C_{i+1}) está presente em E(G).
2.2 Se a aresta não está presente, REJEITE.
3. Se todas as arestas estão presentes em G, ACEITE."

Como a realização de testes e do *loop* (2) são operações realizadas em tempo polinomial, o verificador é polinomial. Logo, HAM-CYCLE ∈ NP.

■

.....

Exercício 7 Mostre que se ***P*** = ***NP***, então existe um algoritmo polinomial que, dado um grafo *G*, **encontra** em *G* um clique de tamanho máximo.

RESPOSTA =

Vamos provar que CLIQUE ∈ NP. Para isso, vamos construir um verificador polinomial para a linguagem CLIQUE.

V = “Com entrada <G,K,C>, onde G é um grafo, V(G) é seu conjunto de vértices e E(G) é seu conjunto de arestas. C é o conjunto de vértices a ser verificado e K é o tamanho do clique.

1. Para todo v Є C, verifique se v possui arestas aos demais vértices de C.
2. Se verdadeiro, então:

2.1 Se |C| ≤ K, ACEITE.

2.2 Senão, REJEITE.

3. REJEITE.”

Como a realização de comparações e testes corresponde a operações em tempo polinomial, o verificador é polinomial. Logo, CLIQUE Є NP.

Se P = NP e CLIQUE Є NP, temos que CLIQUE Є P. Então, existe uma M.T. determinística M que decide CLIQUE em tempo polinomial, onde M recebe como entrada um grafo G e um número K, ou seja, (<G,K>).

Vamos construir uma M.T. N que encontra em G um clique de tamanho máximo:

N = “Com entrada <G>:

1. K = 0.
2. Para i de 1 até n, onde n é o número de vértices de G, faça:

2.1 Simule M com entrada <G,i>.

2.2 Se M aceita, K = i.

2.3 Se M rejeita, abandone este *loop* “para”.

3. Para i de 1 até n, onde n é o número de vértices de G, faça:

3.1 “Arranque” o vértice i de G.

3.2 Simule M com entrada <G,K>.

3.3 Se M rejeita, “devolva” o vértice i ao grafo G.

4. Retorne <G,K>.”

■

.....

Obs1: Uma coloração própria de um grafo G é uma atribuição de cores aos vértices tal que nenhuma aresta possui as extremidades com a mesma cor. Ciclo hamiltoniano é um ciclo que passa por todos os vértices do grafo.